(NeXT Tip #17) NXClose vs. NXCloseMemory

Christopher Lane (*lane[at]CAMIS.Stanford.EDU*)
Fri, 29 Jan 1993 11:19:56 -0800 (PST)

(Yes, I realize this is the second tip this week but I got bit by this and looking around at *.m files on CAMIS, others are mixing up these routines.)

Contrary to what some sections of the NeXT documentation might imply (or outright state), you should use NXCloseMemory() with the NXMapFile() and NXOpenMemory() routines, never NXClose(). The NXClose() routine should be used with the NXOpenFile() routine only!

What's the difference? If you use NXMapFile(), for example, to open a NX_READONLY stream onto a file, mapping that file into virtual memory, then if you close it via NXClose(), you will leave the virtual memory mapped. This isn't a big deal if you just open a file once, read it and close it in your application as memory is reclaimed when your it terminates. However, if you're opening multiple files (e.g. in a loop), this can eat up your virtual memory and/or swap space very quickly. So, it's a bad habit to get into and you should avoid it.

The NeXT is clever about handing out real and swap memory so it is possible to allocate far more virtual memory than the total of RAM, swap or even disk as long as you don't try to use it -- when you do, it then requires allocation of real resources. So, in some cases, you can incorrectly use NXClose(), leave memory mapped, and never pay a penalty. But the next time, when you use the stream in a slightly different way, you might be in serious trouble.

There are situations, typically when using NX_WRITEONLY, where it's legal to use NXClose() on a stream created via NXMapFile() or NXOpenMemory() since it will do the equivalent of a NXCloseMemory(stream, NX_TRUNCATEBUFFER) -- I still recommend you use the (longer) NXCloseMemory() form instead so you don't get into a habit of calling NXClose().

As far as the second argument to NXCloseMemory:

Typically, NX_FREEBUFFER will be used as the second argument to free
all memory used by the stream, but there are two other constants
available. If you've used the stream for writing, more memory may
have been made available than was actually used; the constant
NX_TRUNCATEBUFFER indicates that any unused pages of memory should be
freed. .... NX_SAVEBUFFER doesn't free the memory that had been made
available.

- .../NextDev/GeneralRef/03_Common/Functions/CommonFuncs.rtf

The NX_TRUNCATEBUFFER and NX_SAVEBUFFER arguments are useful if you want to open a stream onto memory, write to it, and further manipulate the memory after the stream is closed.

Below are example code fragments using the various NXStream open and close routines. (For the list of *.m files I found on CAMIS using NXClose() where NXCloseMemory() might be preferable, just drop me a note.) There is also a separate issue, when writing NeXT code, of using the NX*() routines vs. standard Unix fopen() & fclose() calls which I won't get into here.

- Christopher

```
- method:(const char *) filename
{
    NXStream *stream;

    if((stream = NXMapFile(filename, NX_READONLY)) != NULL) {
            ...
            NXCloseMemory(stream, NX_FREEBUFFER);
            }
    else (void) fprintf(stderr, " ... \n", filename);

    return self;
}

- method:(const char *) filename
{
    int fd;
    NXStream *stream;

    if ((fd = open(filename, O_RDONLY, 0644)) != CERROR) {
        if ((stream = NXOpenFile(fd, NX_READONLY)) != NULL) {
            ...
            NXClose(stream);
            }
        else (void) fprintf(stderr, " ... \n", filename);
        ...
        (void) close(fd);
        }
    else (void) fprintf(stderr, " ... \n", filename);

    return self;
}


- method:(const char *) string
{
    char *memory;
    NXStream *stream;
    vm_size_t size = strlen(string);
```

```
    (void) vm_allocate(task_self(), (vm_address_t *) &memory, size, TRUE);

    (void) strcpy(memory, string);

    if((stream = NXOpenMemory(memory, size, NX_READONLY)) != NULL) {
        ...
        NXCloseMemory(stream, NX_FREEBUFFER);
        }
    else (void) fprintf(stderr, " ... \n", string);

    return self;
}
```